Introduction to RISC-V Processor Verification Methodology With Dynamic Testbench For Asynchronous Events

SemIsrael Tech Webinar

Larry Lapides

22 February 2022

Agenda

- Introduction to Imperas
- Why RISC-V?
- RISC-V processor design verification (DV) issues
- 5 levels of RISC-V DV methodology
- Key technologies: reference models, verification IP
- Summary

Agenda

- **Introduction to Imperas**
- Why RISC-V?
- RISC-V processor design verification (DV) issues
- 5 levels of RISC-V DV methodology
- Key technologies: reference models, verification IP
- Summary

Imperas Founding Story

- Imperas founding team has background in Electronic Design Automation (EDA) tools, and FPGA and processor IP companies
- Imperas founding team saw the need for tools and methodology similar to EDA for software debug, test and analysis, based on software simulation
- Just as no SoC is developed without significant simulation, and with verification tools on top of simulation, we believe the embedded community is evolving so that no embedded system will be shipped without significant simulation-based verification of software
- Key Imperas differentiation: Imperas products have been architected from the tool requirements down, not from the modeling requirements up
- With the introduction and adoption of RISC-V, Imperas has added DV technology and methodology to the portfolio

Imperas and RISC-V

- Q2 2016: Design Automation Conference – Imperas first learns about RISC-V – looks academic and fragmented
- Q4 2016: RISC-V Workshop (@ Google) – 350 attendees from serious companies, and the ISA looks to be converging
- Q1 2017: Imperas joins the RISC-V Foundation; build first RISC-V processor model
- Q3 2017: Imperas starts participating in the Compliance Working Group; builds/donates ISS and tests
- Q1 2018: Imperas introduces methodology for adding/optimizing custom instructions for RISC-V cores
- Q2 2018: First paying customer using Imperas RISC-V models for software development and design verification (DV)
- Q1 2019: First tape out of RISC-V SoC based on using Imperas model as DV reference model
- Q2 2019: Imperas starts collaborating with Google on DV flows with instruction stream generator
- Q1 2020: Imperas starts working with the OpenHW Group and individual members on DV of Core-V cores
- Q1 2021: Imperas presents 2 papers on RISC-V processor DV at the DVCon Silicon Valley conference (with OpenHW, Nvidia Networking)
- Q4 2021: Imperas introduces ImperasDV RISC-V verification product line

Imperas RISC-V Customers and Partners

The most complex RISC-V processor projects use Imperas

Users

- Nagravision
- Nvidia Networking (Mellanox)
- EM Micro US
- Silicon Labs
- Dolphin Design
- lowRISC (Ibex)
- RISC-V processor IP vendor
- Top-tier systems company (AI application)
- Top-tier consumer electronics company (AR/VR application)
- NSITEXE (DENSO subsidiary)
- Startup building accelerator based on multiprocessor RV64
- Japanese government projects "TRASIO" and "RVSPF"
- Barcelona Supercomputing Center
- Numerous universities around the world

Partners

- RISC-V Intl (compliance, bitmanip, crypto, various events)
- OpenHW (verification working group)
- CHIPS Alliance (verification working group)
- Google (for open source ISG, & co-author DV papers)
- Valtrix (test generation tools)
- Andes (processor IP vendor)
- SiFive (processor IP vendor)
- Codasip (processor IP vendor)
- MIPS (processor IP vendor)

Agenda

- Introduction to Imperas
- **Why RISC-V?**
- RISC-V processor design verification (DV) issues
- 5 levels of RISC-V DV methodology
- Key technologies: reference models, verification IP
- Summary

RISC-V History

- RISC-V is an open standard instruction set architecture (ISA) that began in 2010 at the University of California, Berkeley
- Unlike most other ISAs, RISC-V is provided under open source licenses that do not require fees to use
  - This is just the architecture, not the processor implementation
- Unlike other academic designs which are typically optimized only for simplicity of exposition, the designers intended that the RISC-V instruction set be usable for practical computers
- While the ISA is a comprehensive RISC architecture, the RISC-V specification allows for users to add custom features (instructions, CSRs, …)
- **The freedom, and not the free, is why RISC-V usage is growing so fast**

Freedom Enables Domain Specific Processing

- RISC-V is growing in market segments where x86 (PCs, data centers) and Arm (mobile) architectures are not dominant
  - Small microcontrollers for SoC management, replacing proprietary cores
  - Verticals such as IoT and AI/ML
  - Horizontal markets such as security
- The freedom of the open ISA enables users to develop *differentiated* domain specific processors
- RISC-V users include traditional semiconductor companies, and embedded systems companies now practicing vertical integration by developing their own SoCs

Agenda

- Introduction to Imperas
- Why RISC-V?
- **RISC-V processor design verification (DV) issues**
- 5 levels of RISC-V DV methodology
- Key technologies: reference models, verification IP
- Summary

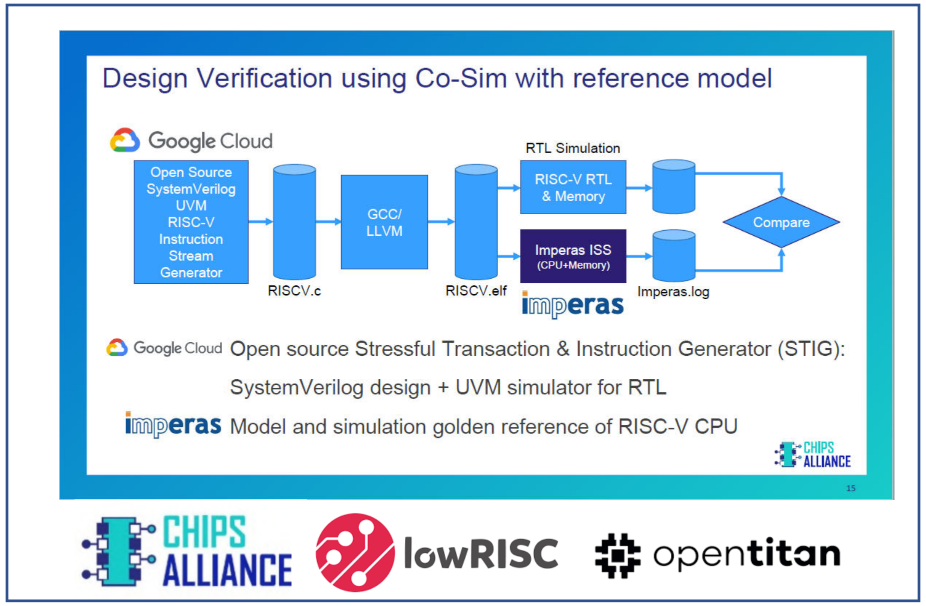
Challenges in RISC-V Processor DV

- Feature selection and design choices require serious consideration due to implications of every decision
  - Experienced processor teams know the costs associated with every feature
    - Every addition dramatically compounds verification complexity
    - Costs of simple added feature can be huge – and unknown to inexperienced teams
    - Adds schedule, resources, quality costs == big risks
- As of 2021, no off-the-shelf toolkit/products available for DV of processors
  - No EDA vendor has 'RISC-V CPU DV kit' product
  - There are in-house proprietary solutions in CPU developers (e.g. Intel, AMD, Arm, …)
  - Building your own processor adds schedule, resources, quality costs … and risks
- Current SoC cost is 50% for HW DV (with CPUs bought in as proven IP)
  - Developing own CPU adds huge DV incremental schedule, resources, quality challenges

Agenda

- Introduction to Imperas
- Why RISC-V?
- RISC-V processor design verification (DV) issues
- **5 levels of RISC-V DV methodology**
  1) **Hello World**
  2) **Self-checking tests (e.g. Berkeley torture tests pre-2018)**
  3) **Post-simulation trace log file compare**
  4) **Synchronous step-and-compare**
  5) **Asynchronous step-and-compare**
- Key technologies: reference models, verification IP
- Summary

3) Post-Simulation Trace Log File Compare (Entry Level DV)

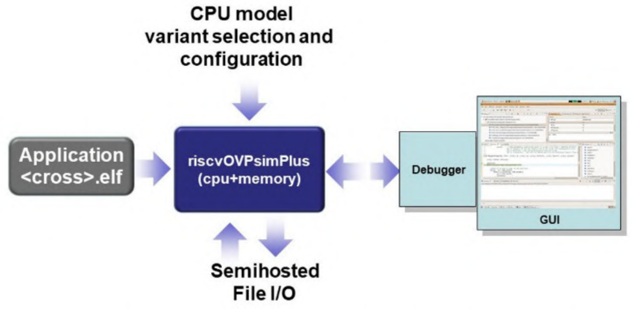


Design Verification using Co-Sim with reference model

Google Cloud Open source Stressful Transaction & Instruction Generator (STIG):

SystemVerilog design + UVM simulator for RTL

imperas Model and simulation golden reference of RISC-V CPU


CPU model variant selection and configuration
Application <cross>.elf
riscvOVPsimPlus (cpu+memory)
Debugger
GUI
Semihosted File I/O

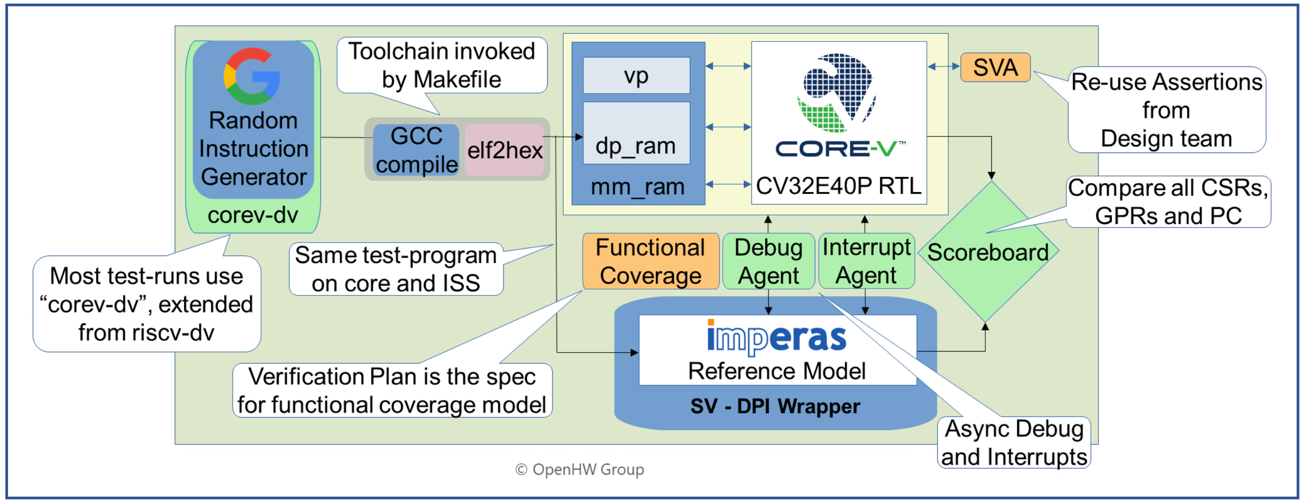
Imperas riscvOVPsimPlus Reference Simulator

- Process
  - use random generator (ISG) to create tests
  - during simulation of ISS write trace log file
  - during simulation of RTL write trace log file
  - at the end of both runs, run logs through compare program to see differences / failures
- ISS: riscvOVPsimPlus includes Trace and GDB interface
  - Free ISS: https://www.ovpworld.org/riscvOVPsimPlus
- ISG: riscv-dv from Google Cloud / Chips Alliance
  - Free ISG: https://github.com/google/riscv-dv

5) Asynchronous Step-Compare (Highest Quality DV Methodology)

- Design features needing this methodology include OoO and multi-issue pipeline, multi-hart processor, debug mode, interrupts, …
  - Example SystemVerilog components
    - tracer: Reports instructions for checking and register writebacks
    - step_and_compare: Manages the reference model and checks functionality
    - interrupt_assert: Properties for interrupt coverage/checking
    - debug_assert: Properties for debug coverage/checking
- Typically hard, complex, and expensive to get working
  - Challenge is extracting async info from micro-architecture RTL pipeline

Example flow:


Random Instruction Generator
corev-dv
Toolchain invoked by Makefile
GCC compile
elf2hex
vp
dp_ram
mm_ram
CORE-V
CV32E40P RTL
SVA
Re-use Assertions from Design team
Compare all CSRs, GPRs and PC
Scoreboard
Functional Coverage
Debug Agent
Interrupt Agent
Most test-runs use "corev-dv", extended from riscv-dv
Same test-program on core and ISS
Verification Plan is the spec for functional coverage model
imperas Reference Model
SV - DPI Wrapper
Async Debug and Interrupts
© OpenHW Group

2nd generation CV32E40P OpenHW flow (2H2020)
(Imperas model encapsulated in SystemVerilog)

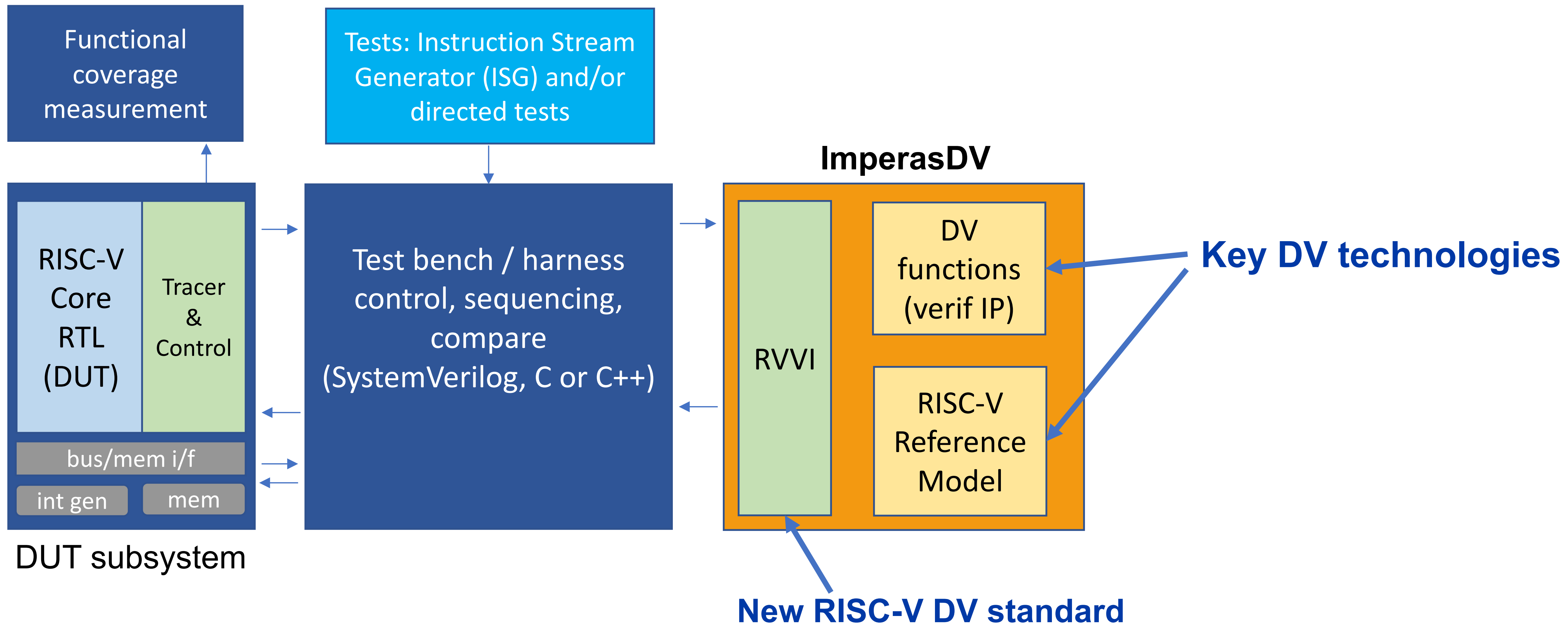
Asynchronous Step-Compare Summary

- Instruction by instruction lockstep comparison (**includes async events**)
  - Comparison of execution flow, of program data, of programmers and internal state
- Immediate comparison
  - Allows for debug introspection at point of failure – very powerful
  - Does not waste execution cycles after failure
- Includes focus on async events, control flow, and hardware real time effects
- Supports multi-hart processors and out-of-order and multi-issue pipelines
- Can be hard to develop & set up (depends on RTL DUT tracer features and pipeline understanding)
- Can be expensive in terms of time, resources, licenses => costs a lot per bug found
  - But the bugs are even more expensive if not found early enough …
- Async step-compare is the most comprehensive and most efficient DV approach
- Next steps for async step-compare: standards for the test bench; verification IP

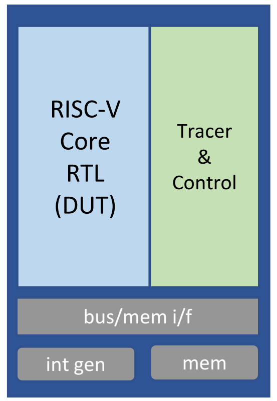
Agenda

- Introduction to Imperas
- Why RISC-V?
- RISC-V processor design verification (DV) issues
- 5 levels of RISC-V DV methodology
- **Key technologies: reference models, verification IP**
- Summary

RISC-V Processor DV Environment has 5 Major Components

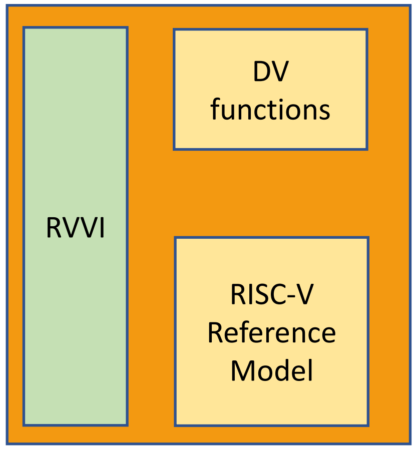


RVVI: RISC-V Verification Interface Standard for Connecting to Test Harness



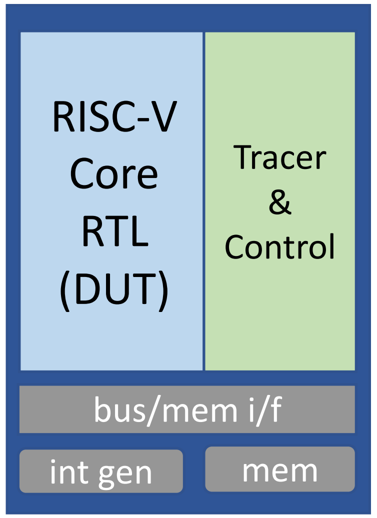
DUT subsystem

ImperasDV



- https://github.com/riscv-verification/RVVI (Public Open Standard)
- RVVI-VLG
  - Verilog DUT interfaces
    - RVVI-VLG state – streaming 'tracer' data
    - RVVI-VLG nets – implementation dependent (Interrupts, Debug)
    - Handles multi-hart, multi-issue, Out-of-Order
- RVVI-API
  - Controls DV subsystem and reference model
  - RVVI_state - RISC-V Verification Interface - State
  - RVVI_control - RISC-V Verification Interface - Control
  - RVVI_io - RISC-V Verification Interface - IO (Interrupts, Debug)
  - RVVI_bus - RISC-V Verification Interface - (Data, Instruction Bus)
  - Supports SystemVerilog, C, C++ testbenches

RTL DUT with Tracer Interface



DUT subsystem

- The key component – the DUT being tested
  - Includes memory model and bus interfaces
  - Includes interrupt generator
- Requires a tracer to provide appropriate data to the test bench
- Requires control interface so test bench can step through events
- ***Quality of the tracer determines the potential capabilities of the DV environment***

RISC-V Model Requirements
Not Just for DV; Also for SW Dev

- Model the ISA, including all versions of the ratified spec, and stable unratified extensions
- Easily update and configure the model for the next project
- User-extendable for custom instructions, registers, …
- Model actual processor IP, e.g. Andes, SiFive, OpenHW, Codasip, MIPS, …
- Well-defined test process including coverage metrics
- Interface to other simulators, e.g. SystemVerilog, SystemC, Imperas virtual platform simulators
- Interface to software debug tools, e.g. GDB/Eclipse, Imperas MPD
- Interface to software analysis tools including access to processor internal state, etc.
- Interface to architecture exploration tools including extensibility to timing estimation

- Most RISC-V ISSs can meet one or two of these requirements
- Imperas models and simulators were built to satisfy these requirements, and matured through usage on non-RISC-V ISAs over the last 12+ years

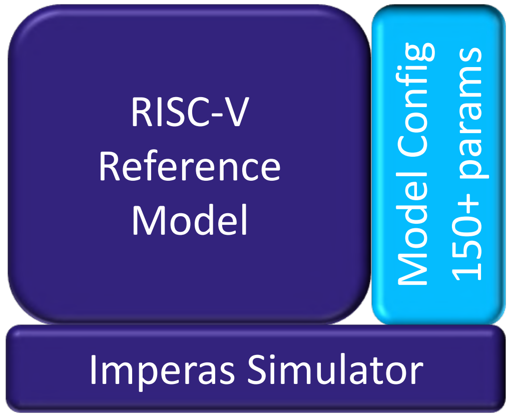
OVP Library of RISC-V Fast Processor Models

- Existing Imperas Open Virtual Platforms (OVP) Fast Processor Models of …
  - Generic or envelope models of RV32/64 IMAFDCEVBHKP M/S/U privilege modes
  - Models of processor IP vendors: Andes, Codasip, MIPS, OpenHW, SiFive
  - Custom models for users building their own RISC-V processors
- Custom instructions easily added by user or by Imperas
  - New instructions are added in side file so as not to perturb the verified model
    - Imperas tools work with the complete processor model, including the custom instructions
  - Custom instructions can be analyzed for effectiveness using instruction coverage, profiling tools
  - Video demo: http://www.imperas.com/risc-v-custom-instruction-design-and-verification-flow
- Models are built using Test Driven Development (TDD) methodology
  - Tests are built at the same time as features are added
  - Continuous Integration (CI) test flow used
  - > 15,000 directed tests for models + simulator
  - Additional testing by processor IP vendors to validate models

"The Imperas virtual platform solutions for software development, debug and test, along with their open-source models, will help accelerate SoC and embedded software development for our customers."
Charlie Hong-Men Su, Ph.D., Andes Technology CTO

Open Virtual Platforms

Imperas is the Reference Model

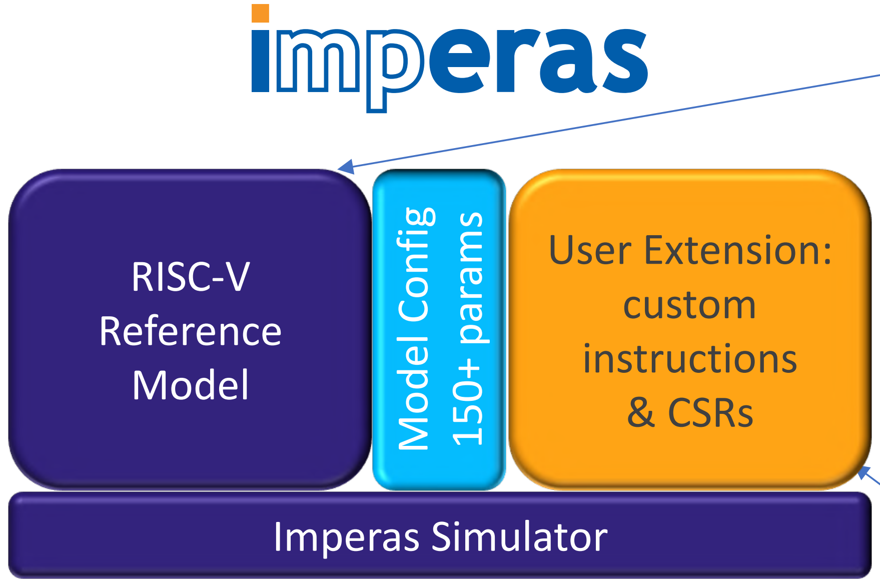


http://www.imperas.com/riscv

- Imperas provides full RISC-V Specification envelope model
- Industrial quality model /simulator of RISC-V processors for use in compliance, verification and test development
- Complete, fully functional, configurable model / simulator
  - All 32bit and 64bit features of ratified User and Privilege RISC-V specs
  - Vector extension, versions 0.7.1, 0.8, 0.9, 1.0
  - Bit Manipulation extension, versions 0.91, 0.92. 0.93, 1.0.0
  - Hypervisor version 0.6.1
  - K-Crypto Scalar version 0.7.1, 1.0.0
  - Debug versions 0.13.2, 0.14, 1.0.0
- Model source included under Apache 2.0 open source license
- Used as reference by :
  - Mellanox/Nvidia, Seagate, NSITEXE/Denso, Google Cloud, Chips Alliance, lowRISC, OpenHW Group, Andes, Valtrix, SiFive, Codasip, MIPS, Nagra/Kudelski, Silicon Labs, RISC-V Compliance Working Group, …

Imperas is used as RISC-V Golden Reference Model

Imperas Model Extensibility



- Separate source files and no duplication to ensure easy maintenance
- Imperas or user can develop the extension
- User extension source can be proprietary

Imperas develops and maintains base model

- Base model implements RISC-V specification in full
- Fully user configurable to select which ISA extensions
- Fully user configurable to select which version of each ISA extension
  - Updated very regularly as ISA extension specification versions change
- Fully user configurable for all RISC-V specification options
  - e.g. implemented optional CSRs, read only or read/write bits etc…

Imperas provides methodology to easily extend base model

- Templates to add new instructions
- Code fragment for adding functionality
- 100+ page user guide/reference manual with many examples
  - Includes example extended processor model

Imperas model is architected for easy extension & maintenance

Flow to Add New Custom Instructions

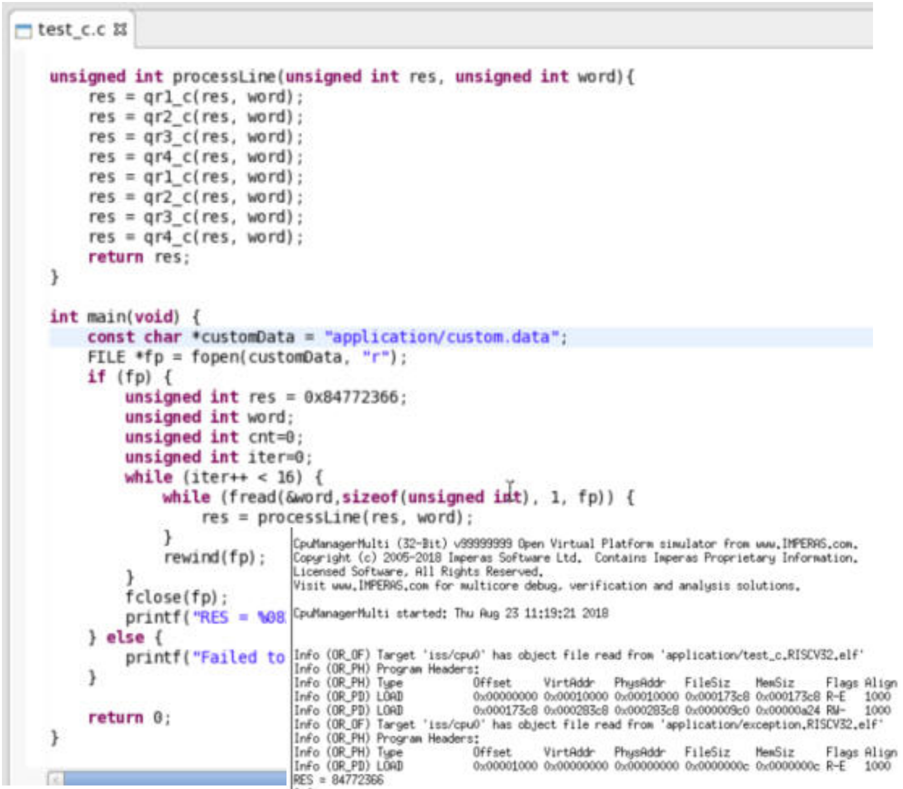
Characterize C Application

- Simulation
- Trace / Debug
- Function Profiling

Simulation of C Application

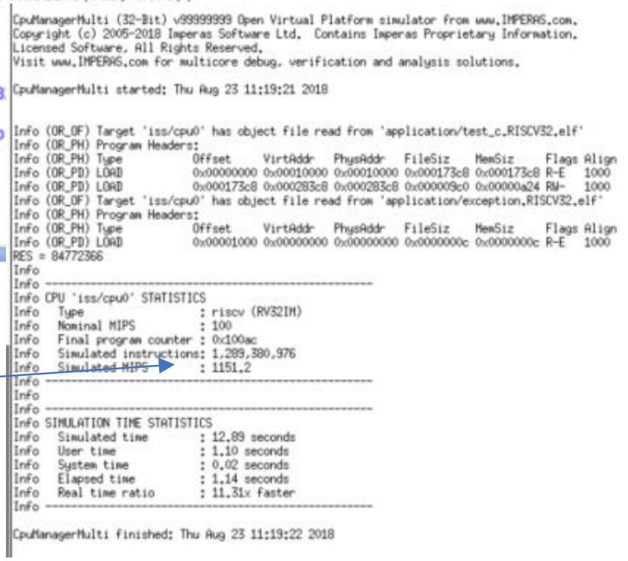
- Cross compiled C application targeting RV32IM
  - Character stream encoder, with ChaCha20 encryption algorithm
- IA simulation
  - Imperas RISC-V ISS with configurable model of RISC-V specification selecting RV32IM
- Semihosting
  - Enables bare metal application to very simply access host I/O

➢ runs fast
  - Over 1 billion instructions a second (standard PC)
    - Linux and Windows supported host OS



test_c.c

```c
unsigned int processLine(unsigned int res, unsigned int word){
    res = qr1_c(res, word);
    res = qr2_c(res, word);
    res = qr3_c(res, word);
    res = qr4_c(res, word);
    res = qr1_c(res, word);
    res = qr2_c(res, word);
    res = qr3_c(res, word);
    res = qr4_c(res, word);
    return res;
}

int main(void) {
    const char *customData = "application/custom.data";
    FILE *fp = fopen(customData, "r");
    if (fp) {
        unsigned int res = 0x84772366;
        unsigned int word;
        unsigned int cnt=0;
        unsigned int iter=0;
        while (iter++ < 16) {
            while (fread(&word,sizeof(unsigned int), 1, fp)) {
                res = processLine(res, word);
            }
            rewind(fp);
        }
        fclose(fp);
        printf("RES = %08
    } else {
        printf("Failed to
    }

    return 0;
}
```

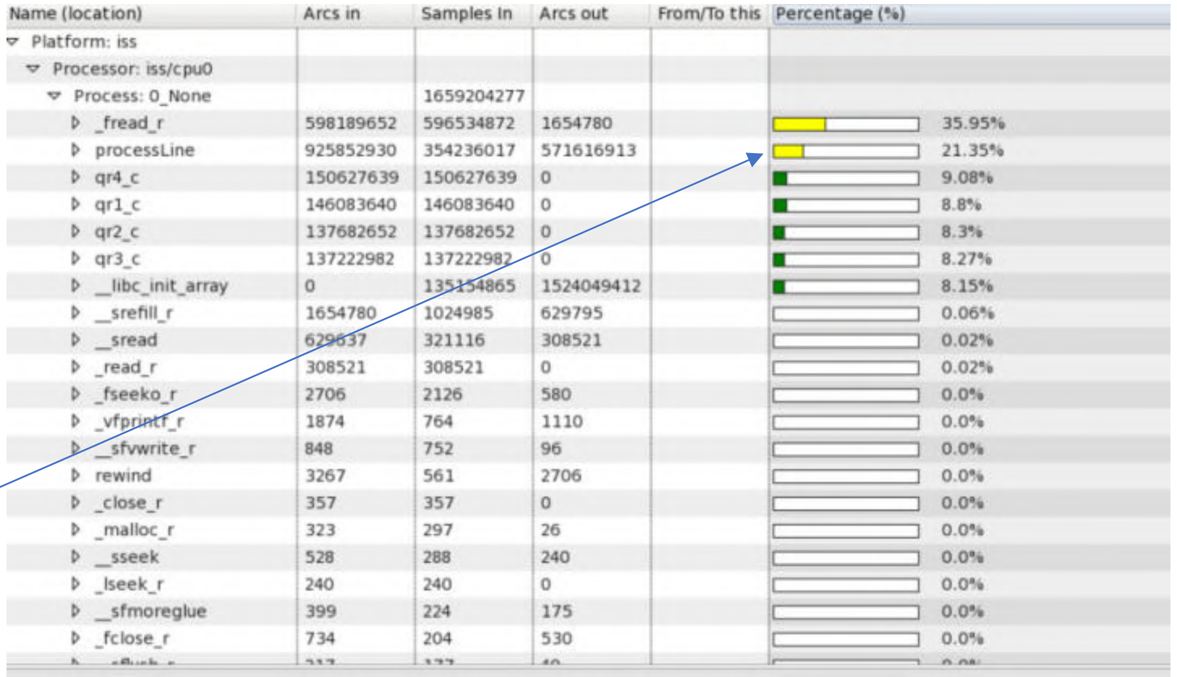


```
CpuManagerMulti (32-Bit) v99999999 Open Virtual Platform simulator from www.IMPERAS.com.
Copyright (c) 2005-2018 Imperas Software Ltd.  Contains Imperas Proprietary Information.
Licensed Software, All Rights Reserved.
Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.

CpuManagerMulti started: Thu Aug 23 11:19:21 2018


Info (OR_OF) Target 'iss/cpu0' has object file read from 'application/test_c.RISCV32.elf'
Info (OR_PH) Program Headers:
Info (OR_PH) Type          Offset     VirtAddr   PhysAddr   FileSiz    MemSiz     Flags Align
Info (OR_PD) LOAD          0x00000000 0x00010000 0x00010000 0x000173c8 0x000173c8 R-E   1000
Info (OR_PD) LOAD          0x000173c8 0x000283c8 0x000283c8 0x000009c0 0x00000a24 RW-   1000
Info (OR_OF) Target 'iss/cpu0' has object file read from 'application/exception.RISCV32.elf'
Info (OR_PH) Program Headers:
Info (OR_PH) Type          Offset     VirtAddr   PhysAddr   FileSiz    MemSiz     Flags Align
Info (OR_PD) LOAD          0x00001000 0x00000000 0x00000000 0x0000000c 0x0000000c R-E   1000
RES = 84772366
Info
Info ---------------------------------------------------
Info CPU 'iss/cpu0' STATISTICS
Info   Type                  : riscv (RV32IM)
Info   Nominal MIPS          : 100
Info   Final program counter : 0x100ac
Info   Simulated instructions: 1,289,380,976
Info   Simulated MIPS        : 1151.2
Info ---------------------------------------------------
Info
Info ---------------------------------------------------
Info SIMULATION TIME STATISTICS
Info   Simulated time        : 12.89 seconds
Info   User time             : 1.10 seconds
Info   System time           : 0.02 seconds
Info   Elapsed time          : 1.14 seconds
Info   Real time ratio       : 11.31x faster
Info ---------------------------------------------------

CpuManagerMulti finished: Thu Aug 23 11:19:22 2018
```

Function Profile C Application

- Same C application
- IA+E simulation
- Sampled profiling with call stack analysis

➢ Shows proportion of time spent in each application function
  - ➢ 21.35% spent in processLine



Name (location)	Arcs in	Samples In	Arcs out	From/To this	Percentage (%)
▿ Platform: iss					
▿ Processor: iss/cpu0					
▿ Process: 0_None		1659204277			
▹ _fread_r	598189652	596534872	1654780		35.95%
▹ processLine	925852930	354236017	571616913		21.35%
▹ qr4_c	150627639	150627639	0		9.08%
▹ qr1_c	146083640	146083640	0		8.8%
▹ qr2_c	137682652	137682652	0		8.3%
▹ qr3_c	137222982	137222982	0		8.27%
▹ __libc_init_array	0	135154865	1524049412		8.15%
▹ __srefill_r	1654780	1024985	629795		0.06%
▹ __sread	629637	321116	308521		0.02%
▹ _read_r	308521	308521	0		0.02%
▹ _fseeko_r	2706	2126	580		0.0%
▹ _vfprintf_r	1874	764	1110		0.0%
▹ __sfvwrite_r	848	752	96		0.0%
▹ rewind	3267	561	2706		0.0%
▹ _close_r	357	357	0		0.0%
▹ _malloc_r	323	297	26		0.0%
▹ __sseek	528	288	240		0.0%
▹ _lseek_r	240	240	0		0.0%
▹ __sfmoreglue	399	224	175		0.0%
▹ _fclose_r	734	204	530		0.0%

Flow to Add New Custom Instructions

Characterize C Application

- Simulation
- Trace / Debug
- Function Profiling

→

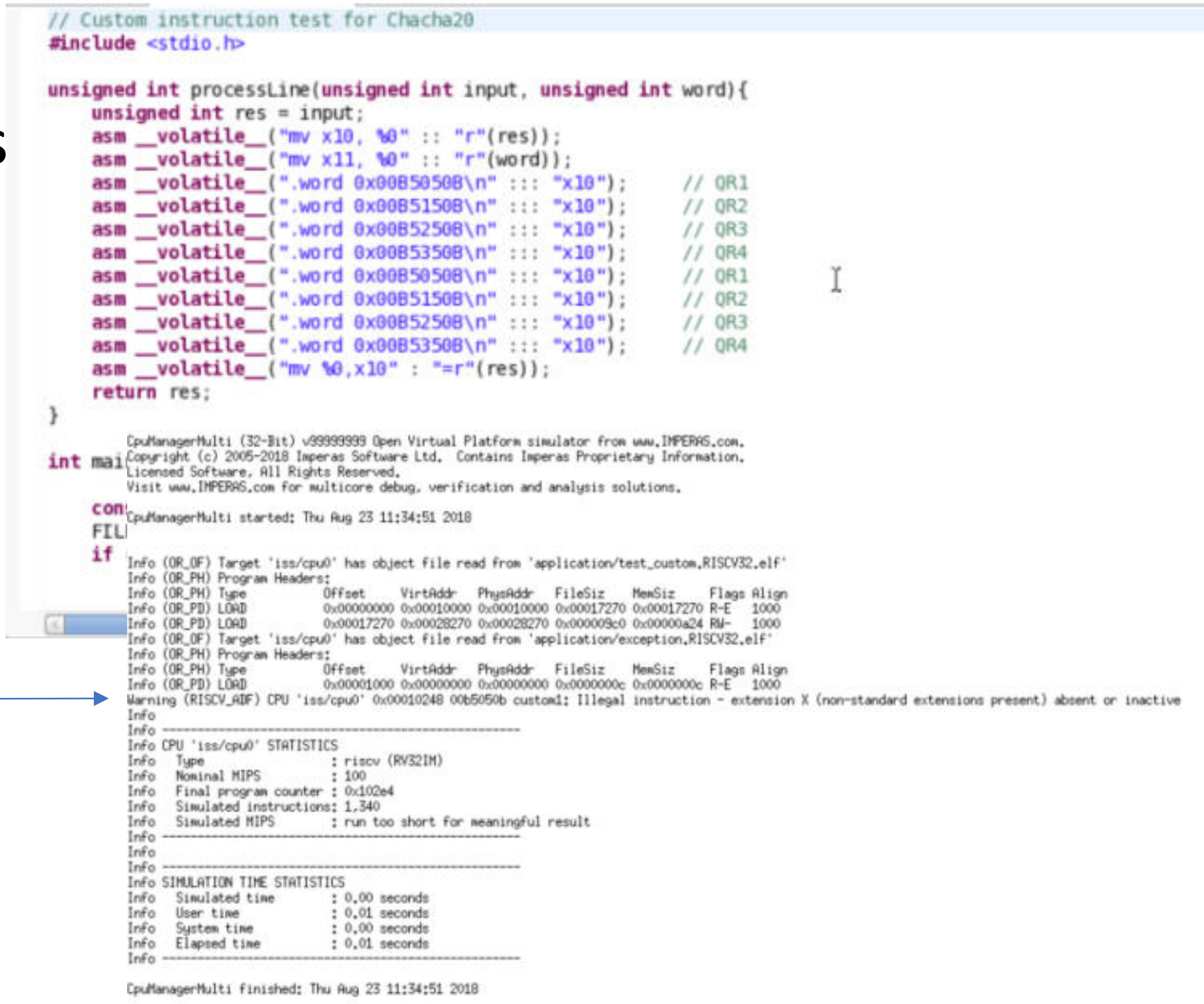
Develop New Custom Instructions

- Design Instructions
- Add to Application
- Add to Model

Add Custom Instructions to Application

- Inline assembly using new instructions replacing C code
- 4 new instructions
- Cross compile using standard tool
- Run on IA simulator

➢ When simulate: unimplemented instruction exception
  - As the instructions have not yet been added to the simulator model



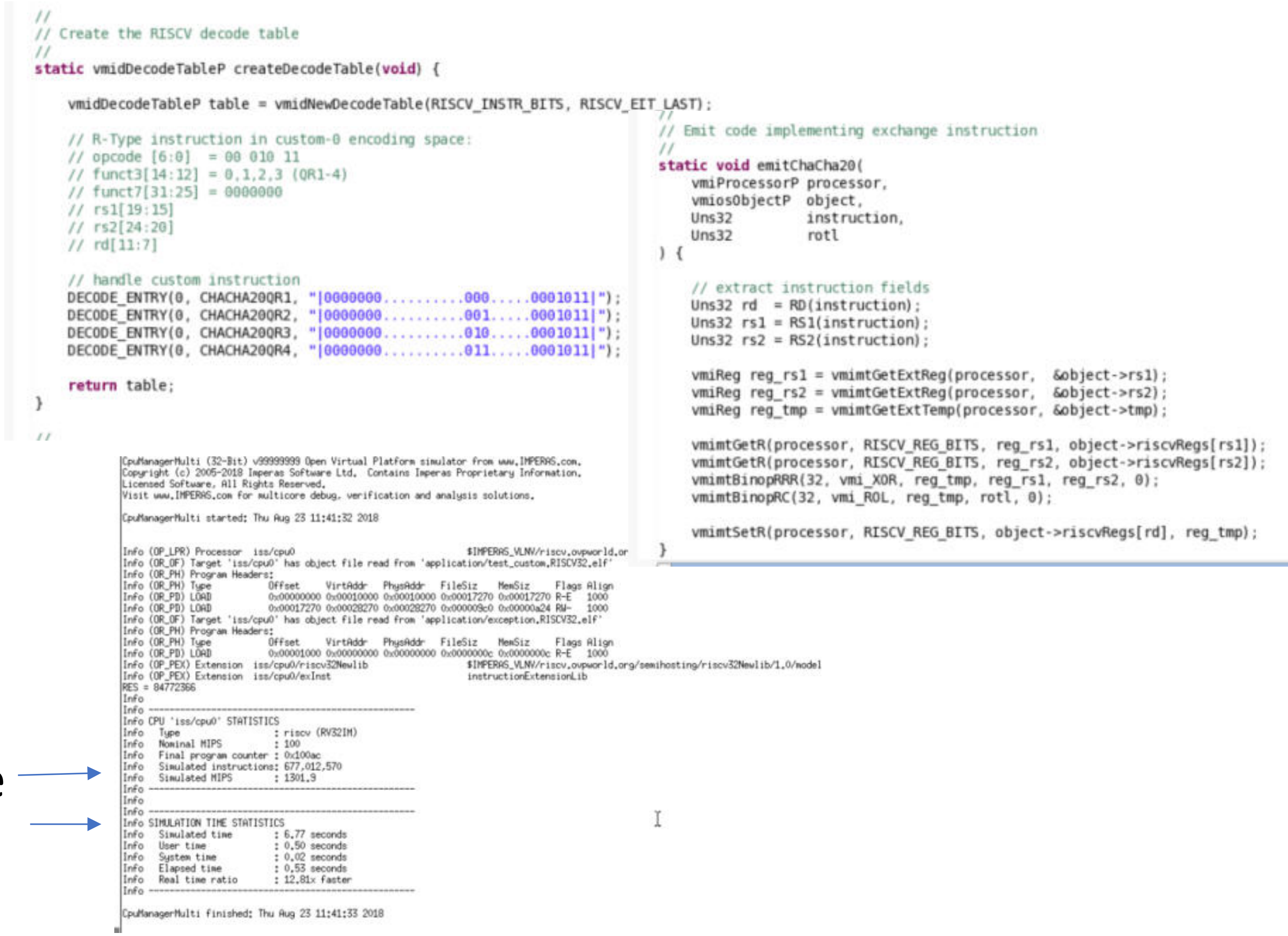
```
// Custom instruction test for Chacha20
#include <stdio.h>

unsigned int processLine(unsigned int input, unsigned int word){
    unsigned int res = input;
    asm __volatile__("mv x10, %0" :: "r"(res));
    asm __volatile__("mv x11, %0" :: "r"(word));
    asm __volatile__(".word 0x00B5050B\n" ::: "x10");      // QR1
    asm __volatile__(".word 0x00B5150B\n" ::: "x10");      // QR2
    asm __volatile__(".word 0x00B5250B\n" ::: "x10");      // QR3
    asm __volatile__(".word 0x00B5350B\n" ::: "x10");      // QR4
    asm __volatile__(".word 0x00B5050B\n" ::: "x10");      // QR1
    asm __volatile__(".word 0x00B5150B\n" ::: "x10");      // QR2
    asm __volatile__(".word 0x00B5250B\n" ::: "x10");      // QR3
    asm __volatile__(".word 0x00B5350B\n" ::: "x10");      // QR4
    asm __volatile__("mv %0,x10" : "=r"(res));
    return res;
}

int mai
    con
    FIL
    if
```

```
CpuManagerMulti (32-Bit) v99999999 Open Virtual Platform simulator from www.IMPERAS.com.
Copyright (c) 2005-2018 Imperas Software Ltd.  Contains Imperas Proprietary Information.
Licensed Software, All Rights Reserved.
Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.

CpuManagerMulti started: Thu Aug 23 11:34:51 2018

Info (OR_OF) Target 'iss/cpu0' has object file read from 'application/test_custom.RISCV32.elf'
Info (OR_PH) Program Headers:
Info (OR_PH) Type           Offset     VirtAddr   PhysAddr   FileSiz    MemSiz     Flags Align
Info (OR_PD) LOAD           0x00000000 0x00010000 0x00010000 0x00017270 0x00017270 R-E   1000
Info (OR_PD) LOAD           0x00017270 0x00028270 0x00028270 0x000009c0 0x00000a24 RW-   1000
Info (OR_OF) Target 'iss/cpu0' has object file read from 'application/exception.RISCV32.elf'
Info (OR_PH) Program Headers:
Info (OR_PH) Type           Offset     VirtAddr   PhysAddr   FileSiz    MemSiz     Flags Align
Info (OR_PD) LOAD           0x00001000 0x00000000 0x00000000 0x0000000c 0x0000000c R-E   1000
Warning (RISCV_ADF) CPU 'iss/cpu0' 0x00010248 00b5050b custom1: Illegal instruction - extension X (non-standard extensions present) absent or inactive
Info
Info ---------------------------------------------------
Info CPU 'iss/cpu0' STATISTICS
Info   Type                  : riscv (RV32IM)
Info   Nominal MIPS          : 100
Info   Final program counter : 0x102e4
Info   Simulated instructions: 1,340
Info   Simulated MIPS        : run too short for meaningful result
Info ---------------------------------------------------
Info
Info ---------------------------------------------------
Info SIMULATION TIME STATISTICS
Info   Simulated time        : 0.00 seconds
Info   User time             : 0.01 seconds
Info   System time           : 0.00 seconds
Info   Elapsed time          : 0.01 seconds
Info ---------------------------------------------------

CpuManagerMulti finished: Thu Aug 23 11:34:51 2018
```

Add Custom Instructions to Model and Re-Simulate

- Use standard Open Virtual Platforms (OVP) instruction modeling APIs to add new instructions (and optional state) as ***new extension library***
  - Easy to extend decode table, add efficient behavioral JIT code
  - Optionally can call directly into user's provided C function of behavior
- Compile and link model extension library
- Simulate IA with ISS plus standard model extended with new library

➢ Instruction count and simulated time have been reduced



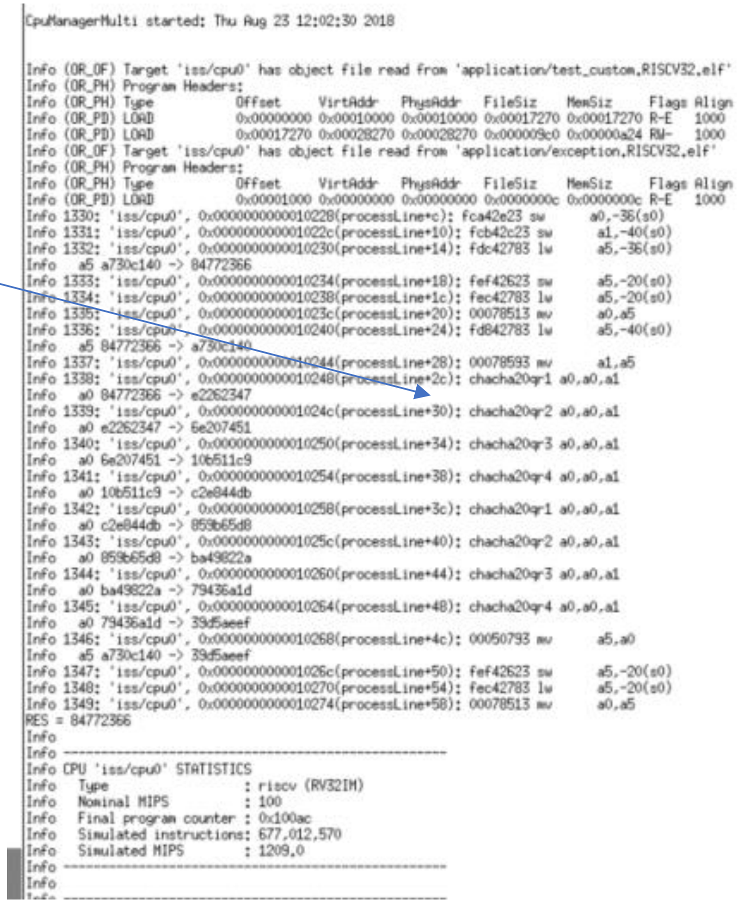
```
//
// Create the RISCV decode table
//
static vmidDecodeTableP createDecodeTable(void) {

    vmidDecodeTableP table = vmidNewDecodeTable(RISCV_INSTR_BITS, RISCV_EIT_LAST);

    // R-Type instruction in custom-0 encoding space:
    // opcode [6:0]   = 00 010 11
    // funct3[14:12] = 0,1,2,3 (QR1-4)
    // funct7[31:25] = 0000000
    // rs1[19:15]
    // rs2[24:20]
    // rd[11:7]

    // handle custom instruction
    DECODE_ENTRY(0, CHACHA20QR1, "|0000000..........000.....0001011|");
    DECODE_ENTRY(0, CHACHA20QR2, "|0000000..........001.....0001011|");
    DECODE_ENTRY(0, CHACHA20QR3, "|0000000..........010.....0001011|");
    DECODE_ENTRY(0, CHACHA20QR4, "|0000000..........011.....0001011|");

    return table;
}
```

```
//
// Emit code implementing exchange instruction
//
static void emitChaCha20(
    vmiProcessorP processor,
    vmiosObjectP  object,
    Uns32         instruction,
    Uns32         rotl
) {
    // extract instruction fields
    Uns32 rd  = RD(instruction);
    Uns32 rs1 = RS1(instruction);
    Uns32 rs2 = RS2(instruction);

    vmiReg reg_rs1 = vmimtGetExtReg(processor,  &object->rs1);
    vmiReg reg_rs2 = vmimtGetExtReg(processor,  &object->rs2);
    vmiReg reg_tmp = vmimtGetExtTemp(processor, &object->tmp);

    vmimtGetR(processor, RISCV_REG_BITS, reg_rs1, object->riscvRegs[rs1]);
    vmimtGetR(processor, RISCV_REG_BITS, reg_rs2, object->riscvRegs[rs2]);
    vmimtBinopRRR(32, vmi_XOR, reg_tmp, reg_rs1, reg_rs2, 0);
    vmimtBinopRC(32, vmi_ROL, reg_tmp, rotl, 0);

    vmimtSetR(processor, RISCV_REG_BITS, object->riscvRegs[rd], reg_tmp);
}
```

```
CpuManagerMulti (32-Bit) v99999999 Open Virtual Platform simulator from www.IMPERAS.com.
Copyright (c) 2005-2018 Imperas Software Ltd.  Contains Imperas Proprietary Information.
Licensed Software, All Rights Reserved.
Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.

CpuManagerMulti started: Thu Aug 23 11:41:32 2018


Info (OP_LPR) Processor  iss/cpu0                          $IMPERAS_VLNV/riscv.ovpworld.or
Info (OR_OF) Target 'iss/cpu0' has object file read from 'application/test_custom.RISCV32.elf'
Info (OR_PH) Program Headers:
Info (OR_PH) Type          Offset     VirtAddr   PhysAddr   FileSiz    MemSiz     Flags Align
Info (OR_PD) LOAD          0x00000000 0x00010000 0x00010000 0x00017270 0x00017270 R-E   1000
Info (OR_PD) LOAD          0x00017270 0x00028270 0x00028270 0x000009c0 0x00000a24 RW-   1000
Info (OR_OF) Target 'iss/cpu0' has object file read from 'application/exception.RISCV32.elf'
Info (OR_PH) Program Headers:
Info (OR_PH) Type          Offset     VirtAddr   PhysAddr   FileSiz    MemSiz     Flags Align
Info (OR_PD) LOAD          0x00001000 0x00000000 0x00000000 0x0000000c 0x0000000c R-E   1000
Info (OP_PEX) Extension  iss/cpu0/riscv32Newlib              $IMPERAS_VLNV/riscv.ovpworld.org/semihosting/riscv32Newlib/1.0/model
Info (OP_PEX) Extension  iss/cpu0/exInst                     instructionExtensionLib
RES = 84772366
Info
Info ---------------------------------------------------
Info CPU 'iss/cpu0' STATISTICS
Info   Type                  : riscv (RV32IM)
Info   Nominal MIPS          : 100
Info   Final program counter : 0x100ac
Info   Simulated instructions: 677,012,570
Info   Simulated MIPS        : 1301.9
Info ---------------------------------------------------
Info
Info ---------------------------------------------------
Info SIMULATION TIME STATISTICS
Info   Simulated time        : 6.77 seconds
Info   User time             : 0.50 seconds
Info   System time           : 0.02 seconds
Info   Elapsed time          : 0.53 seconds
Info   Real time ratio       : 12.81x faster
Info ---------------------------------------------------

CpuManagerMulti finished: Thu Aug 23 11:41:33 2018
```

Trace Custom Instructions

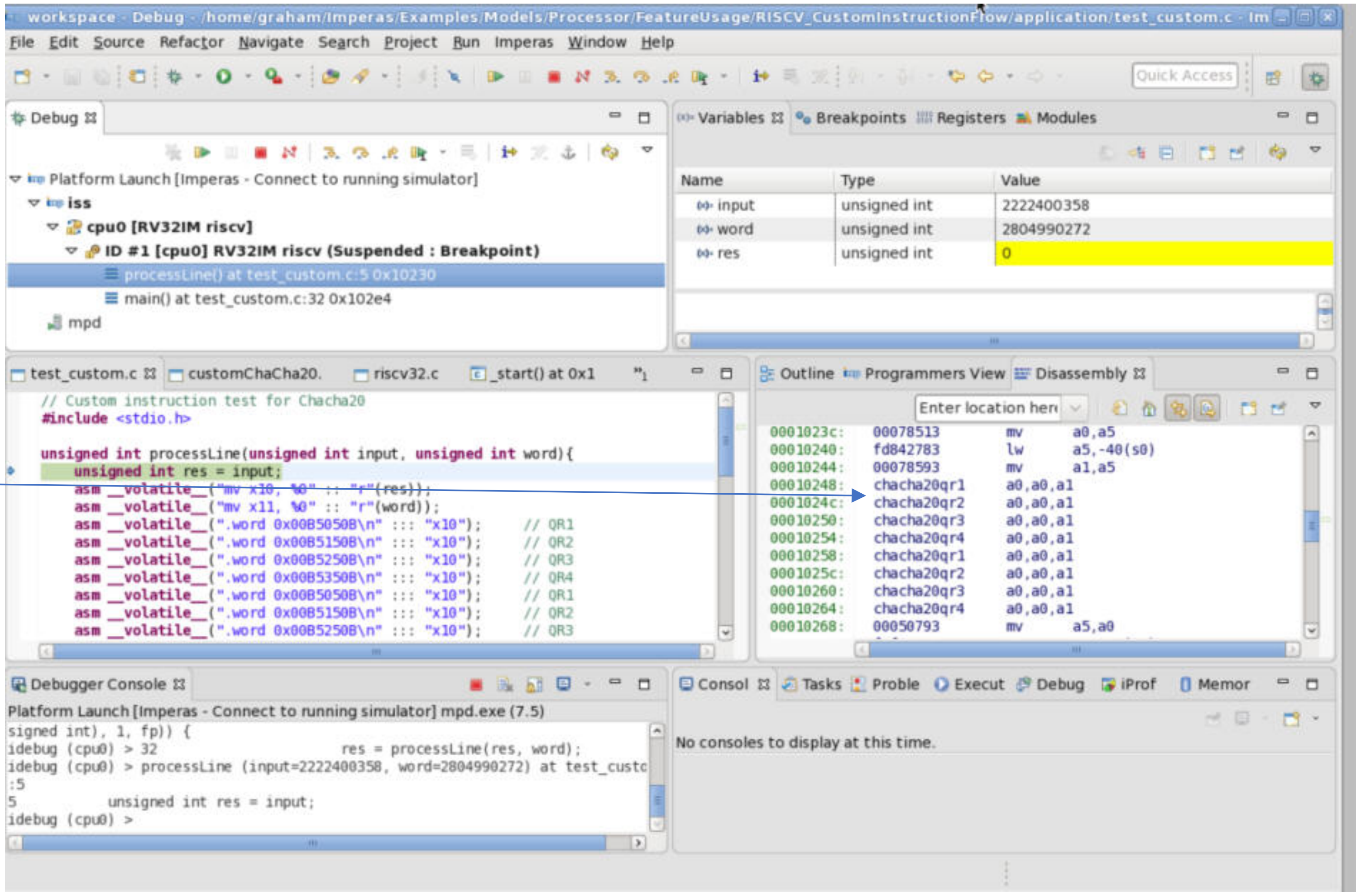
- Simulator has many trace features built in
- See new custom instructions in trace disassembly
- Can select when/where to turn trace on/off
  - Very efficient tracing



```
CpuManagerMulti started: Thu Aug 23 12:02:30 2018


Info (OR_OF) Target 'iss/cpu0' has object file read from 'application/test_custom.RISCV32.elf'
Info (OR_PH) Program Headers:
Info (OR_PH) Type           Offset     VirtAddr   PhysAddr   FileSiz    MemSiz     Flags Align
Info (OR_PD) LOAD           0x00000000 0x00010000 0x00010000 0x00017270 0x00017270 R-E   1000
Info (OR_PD) LOAD           0x00017270 0x00028270 0x00028270 0x000009c0 0x00000a24 RW-   1000
Info (OR_OF) Target 'iss/cpu0' has object file read from 'application/exception.RISCV32.elf'
Info (OR_PH) Program Headers:
Info (OR_PH) Type           Offset     VirtAddr   PhysAddr   FileSiz    MemSiz     Flags Align
Info (OR_PD) LOAD           0x00001000 0x00000000 0x00000000 0x0000000c 0x0000000c R-E   1000
Info 1330: 'iss/cpu0', 0x0000000000010228(processLine+c): fca42e23 sw      a0,-36(s0)
Info 1331: 'iss/cpu0', 0x000000000001022c(processLine+10): fcb42c23 sw      a1,-40(s0)
Info 1332: 'iss/cpu0', 0x0000000000010230(processLine+14): fdc42783 lw      a5,-36(s0)
Info   a5 a730c140 -> 84772366
Info 1333: 'iss/cpu0', 0x0000000000010234(processLine+18): fef42623 sw      a5,-20(s0)
Info 1334: 'iss/cpu0', 0x0000000000010238(processLine+1c): fec42783 lw      a5,-20(s0)
Info 1335: 'iss/cpu0', 0x000000000001023c(processLine+20): 00078513 mv      a0,a5
Info 1336: 'iss/cpu0', 0x0000000000010240(processLine+24): fd842783 lw      a5,-40(s0)
Info   a5 84772366 -> a730c140
Info 1337: 'iss/cpu0', 0x0000000000010244(processLine+28): 00078593 mv      a1,a5
Info 1338: 'iss/cpu0', 0x0000000000010248(processLine+2c): chacha20qr1 a0,a0,a1
Info   a0 84772366 -> e2262347
Info 1339: 'iss/cpu0', 0x000000000001024c(processLine+30): chacha20qr2 a0,a0,a1
Info   a0 e2262347 -> 6e207451
Info 1340: 'iss/cpu0', 0x0000000000010250(processLine+34): chacha20qr3 a0,a0,a1
Info   a0 6e207451 -> 10b511c9
Info 1341: 'iss/cpu0', 0x0000000000010254(processLine+38): chacha20qr4 a0,a0,a1
Info   a0 10b511c9 -> c2e844db
Info 1342: 'iss/cpu0', 0x0000000000010258(processLine+3c): chacha20qr1 a0,a0,a1
Info   a0 c2e844db -> 859b65d8
Info 1343: 'iss/cpu0', 0x000000000001025c(processLine+40): chacha20qr2 a0,a0,a1
Info   a0 859b65d8 -> ba49822a
Info 1344: 'iss/cpu0', 0x0000000000010260(processLine+44): chacha20qr3 a0,a0,a1
Info   a0 ba49822a -> 79436a1d
Info 1345: 'iss/cpu0', 0x0000000000010264(processLine+48): chacha20qr4 a0,a0,a1
Info   a0 79436a1d -> 39d5aeef
Info 1346: 'iss/cpu0', 0x0000000000010268(processLine+4c): 00050793 mv      a5,a0
Info   a5 a730c140 -> 39d5aeef
Info 1347: 'iss/cpu0', 0x000000000001026c(processLine+50): fef42623 sw      a5,-20(s0)
Info 1348: 'iss/cpu0', 0x0000000000010270(processLine+54): fec42783 lw      a5,-20(s0)
Info 1349: 'iss/cpu0', 0x0000000000010274(processLine+58): 00078513 mv      a0,a5
RES = 84772366
Info
Info ---------------------------------------------------
Info CPU 'iss/cpu0' STATISTICS
Info   Type                  : riscv (RV32IM)
Info   Nominal MIPS          : 100
Info   Final program counter : 0x100ac
Info   Simulated instructions: 677,012,570
Info   Simulated MIPS        : 1209.0
Info ---------------------------------------------------
Info
Info ---------------------------------------------------
```

Debug Custom Instructions

- Imperas MPD is Eclipse based source code debug tool
- Can debug using source line or instruction level
- See new custom instructions and any new additional state registers


workspace - Debug - /home/graham/Imperas/Examples/Models/Processor/FeatureUsage/RISCV_CustomInstructionFlow/application/test_custom.c - Im
File Edit Source Refactor Navigate Search Project Run Imperas Window Help
Quick Access
Debug
Variables Breakpoints Registers Modules
test_custom.c customChaCha20. riscv32.c _start() at 0x1
Outline Programmers View Disassembly
Enter location here
Debugger Console
Consol Tasks Proble Execut Debug iProf Memor

```
Platform Launch [Imperas - Connect to running simulator]
  iss
    cpu0 [RV32IM riscv]
      ID #1 [cpu0] RV32IM riscv (Suspended : Breakpoint)
        processLine() at test_custom.c:5 0x10230
        main() at test_custom.c:32 0x102e4
  mpd
```

Name	Type	Value
input	unsigned int	2222400358
word	unsigned int	2804990272
res	unsigned int	0

```
// Custom instruction test for Chacha20
#include <stdio.h>

unsigned int processLine(unsigned int input, unsigned int word){
    unsigned int res = input;
    asm __volatile__("mv x10, %0" :: "r"(res));
    asm __volatile__("mv x11, %0" :: "r"(word));
    asm __volatile__(".word 0x00B5050B\n" ::: "x10");     // QR1
    asm __volatile__(".word 0x00B5150B\n" ::: "x10");     // QR2
    asm __volatile__(".word 0x00B5250B\n" ::: "x10");     // QR3
    asm __volatile__(".word 0x00B5350B\n" ::: "x10");     // QR4
    asm __volatile__(".word 0x00B5050B\n" ::: "x10");     // QR1
    asm __volatile__(".word 0x00B5150B\n" ::: "x10");     // QR2
    asm __volatile__(".word 0x00B5250B\n" ::: "x10");     // QR3
```

```
0001023c:   00078513      mv      a0,a5
00010240:   fd842783      lw      a5,-40(s0)
00010244:   00078593      mv      a1,a5
00010248:   chacha20qr1   a0,a0,a1
0001024c:   chacha20qr2   a0,a0,a1
00010250:   chacha20qr3   a0,a0,a1
00010254:   chacha20qr4   a0,a0,a1
00010258:   chacha20qr1   a0,a0,a1
0001025c:   chacha20qr2   a0,a0,a1
00010260:   chacha20qr3   a0,a0,a1
00010264:   chacha20qr4   a0,a0,a1
00010268:   00050793      mv      a5,a0
```

```
Platform Launch [Imperas - Connect to running simulator] mpd.exe (7.5)
signed int), 1, fp)) {
idebug (cpu0) > 32                          res = processLine(res, word);
idebug (cpu0) > processLine (input=2222400358, word=2804990272) at test_custc
:5
5           unsigned int res = input;
idebug (cpu0) >
```

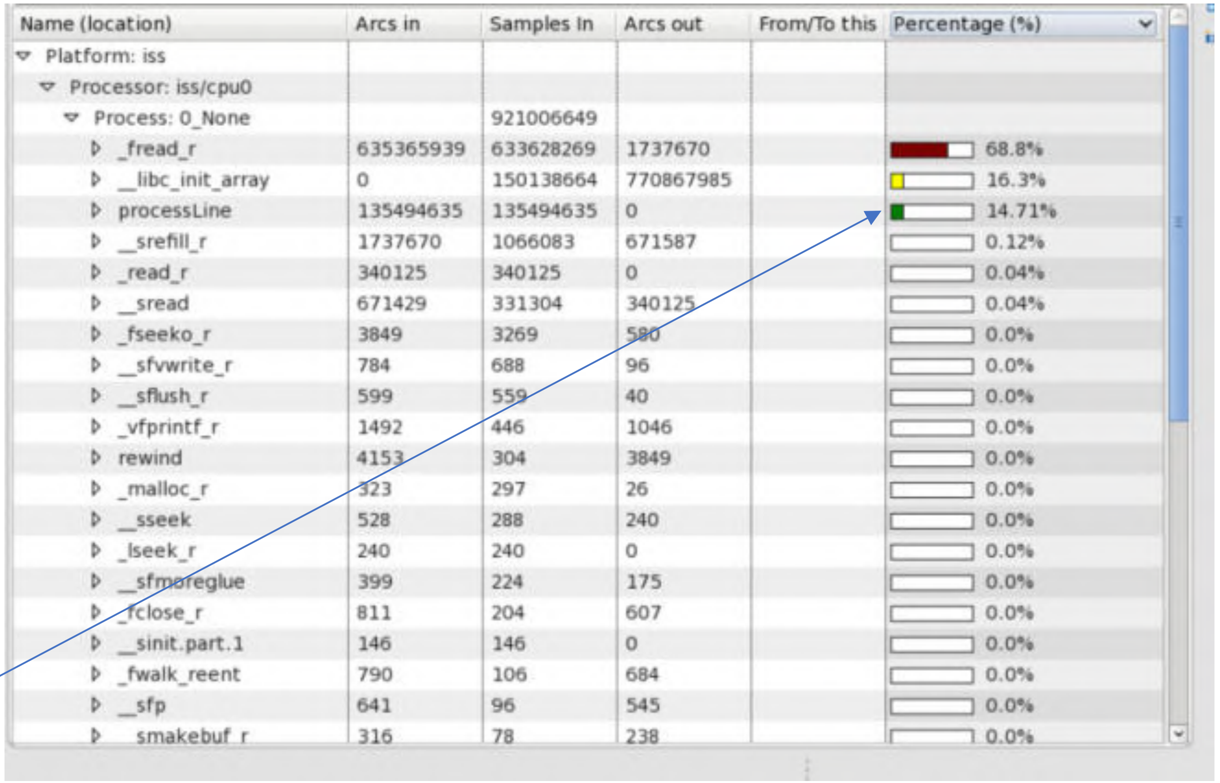
No consoles to display at this time.

Function Profile Application Using Custom Instructions

- IA simulation + timing annotation + custom instructions with sampled profiling

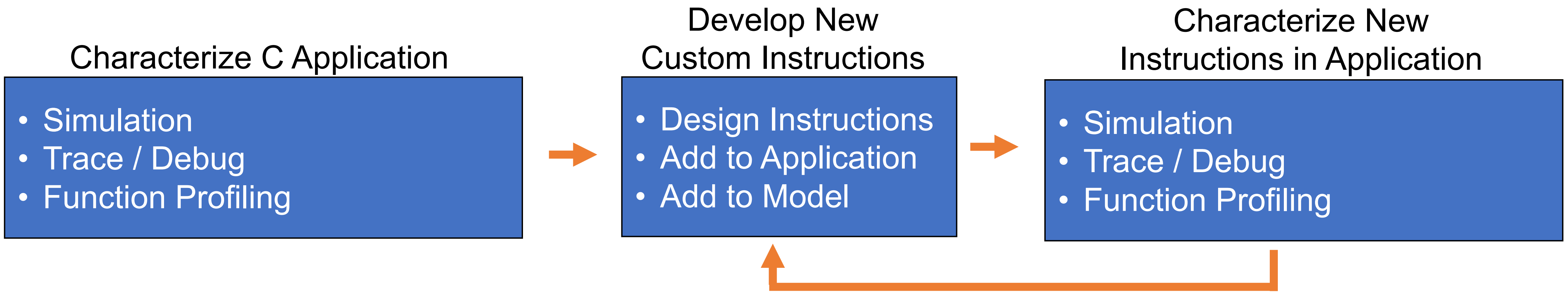
➢ Shows where slowest function is
  - Now much faster…

➢ Shows benefits of using custom instructions
  - ➢ processLine was 21.35% now 14.71%

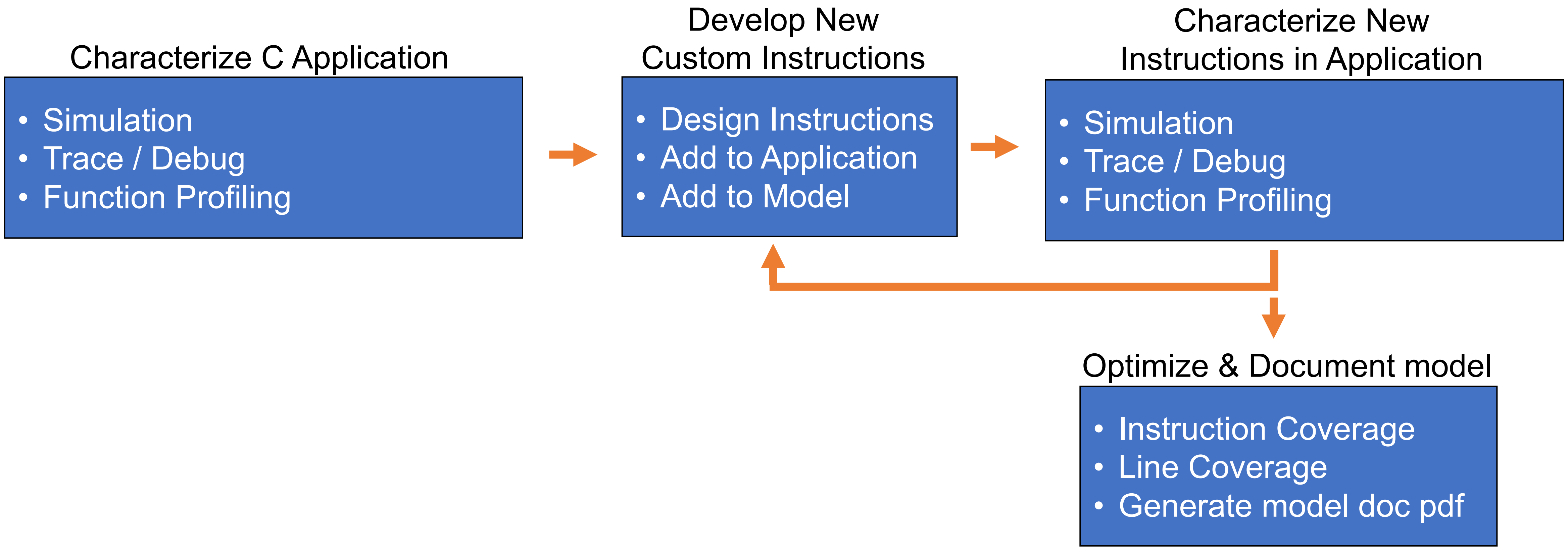


Name (location)	Arcs in	Samples In	Arcs out	From/To this	Percentage (%)
Platform: iss					
Processor: iss/cpu0					
Process: 0_None		921006649			
_fread_r	635365939	633628269	1737670		68.8%
__libc_init_array	0	150138664	770867985		16.3%
processLine	135494635	135494635	0		14.71%
__srefill_r	1737670	1066083	671587		0.12%
_read_r	340125	340125	0		0.04%
__sread	671429	331304	340125		0.04%
_fseeko_r	3849	3269	580		0.0%
__sfvwrite_r	784	688	96		0.0%
__sflush_r	599	559	40		0.0%
_vfprintf_r	1492	446	1046		0.0%
rewind	4153	304	3849		0.0%
_malloc_r	323	297	26		0.0%
__sseek	528	288	240		0.0%
_lseek_r	240	240	0		0.0%
__sfmoreglue	399	224	175		0.0%
_fclose_r	811	204	607		0.0%
__sinit.part.1	146	146	0		0.0%
_fwalk_reent	790	106	684		0.0%
__sfp	641	96	545		0.0%
__smakebuf_r	316	78	238		0.0%

Flow to Add New Custom Instructions



Flow to Add New Custom Instructions



Characterize C Application

- Simulation
- Trace / Debug
- Function Profiling

Develop New Custom Instructions

- Design Instructions
- Add to Application
- Add to Model

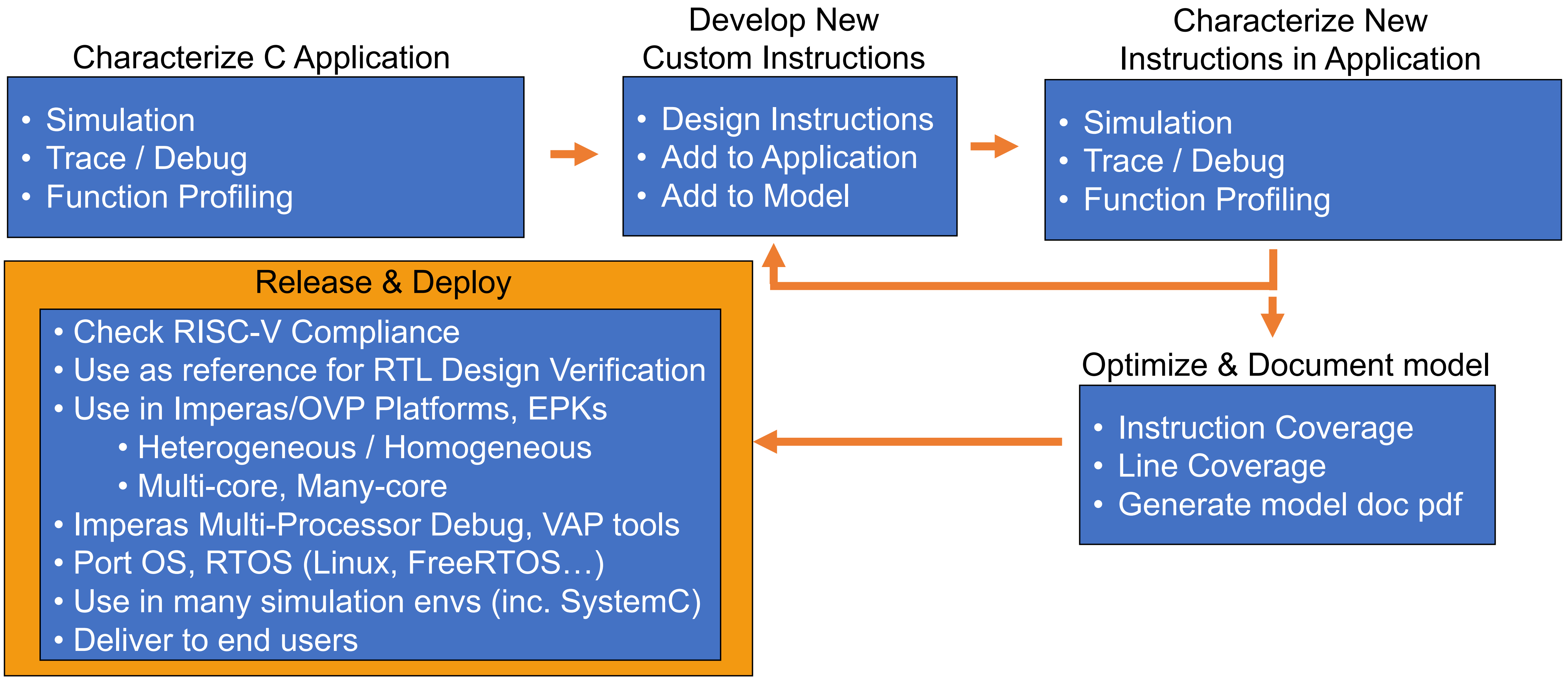
Characterize New Instructions in Application

- Simulation
- Trace / Debug
- Function Profiling

Optimize & Document model

- Instruction Coverage
- Line Coverage
- Generate model doc pdf

Flow to Add New Custom Instructions



Characterize C Application

- Simulation
- Trace / Debug
- Function Profiling

Develop New Custom Instructions

- Design Instructions
- Add to Application
- Add to Model

Characterize New Instructions in Application

- Simulation
- Trace / Debug
- Function Profiling

Optimize & Document model

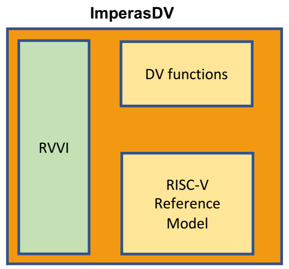
- Instruction Coverage
- Line Coverage
- Generate model doc pdf

Release & Deploy

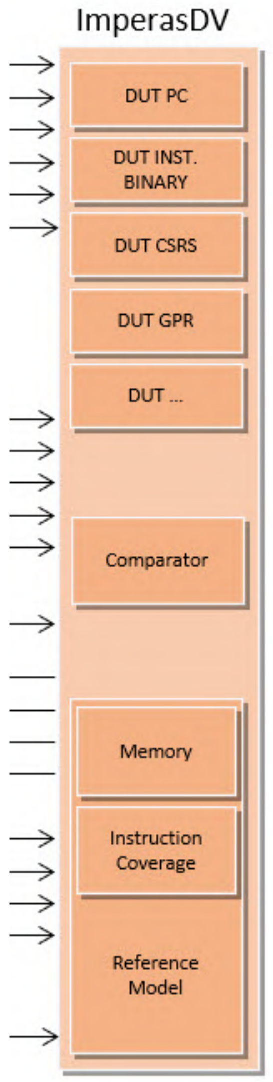
- Check RISC-V Compliance
- Use as reference for RTL Design Verification
- Use in Imperas/OVP Platforms, EPKs
  - Heterogeneous / Homogeneous
  - Multi-core, Many-core
- Imperas Multi-Processor Debug, VAP tools
- Port OS, RTOS (Linux, FreeRTOS…)
- Use in many simulation envs (inc. SystemC)
- Deliver to end users

ImperasDV: RISC-V Verification IP

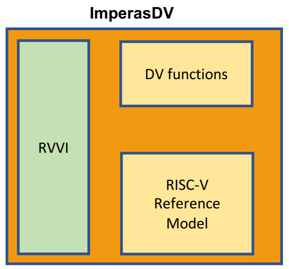
- Verification IP is needed for …
- Ease of use
- Scalability
- Extendability
- Performance
- Debug
- Schedule reduction



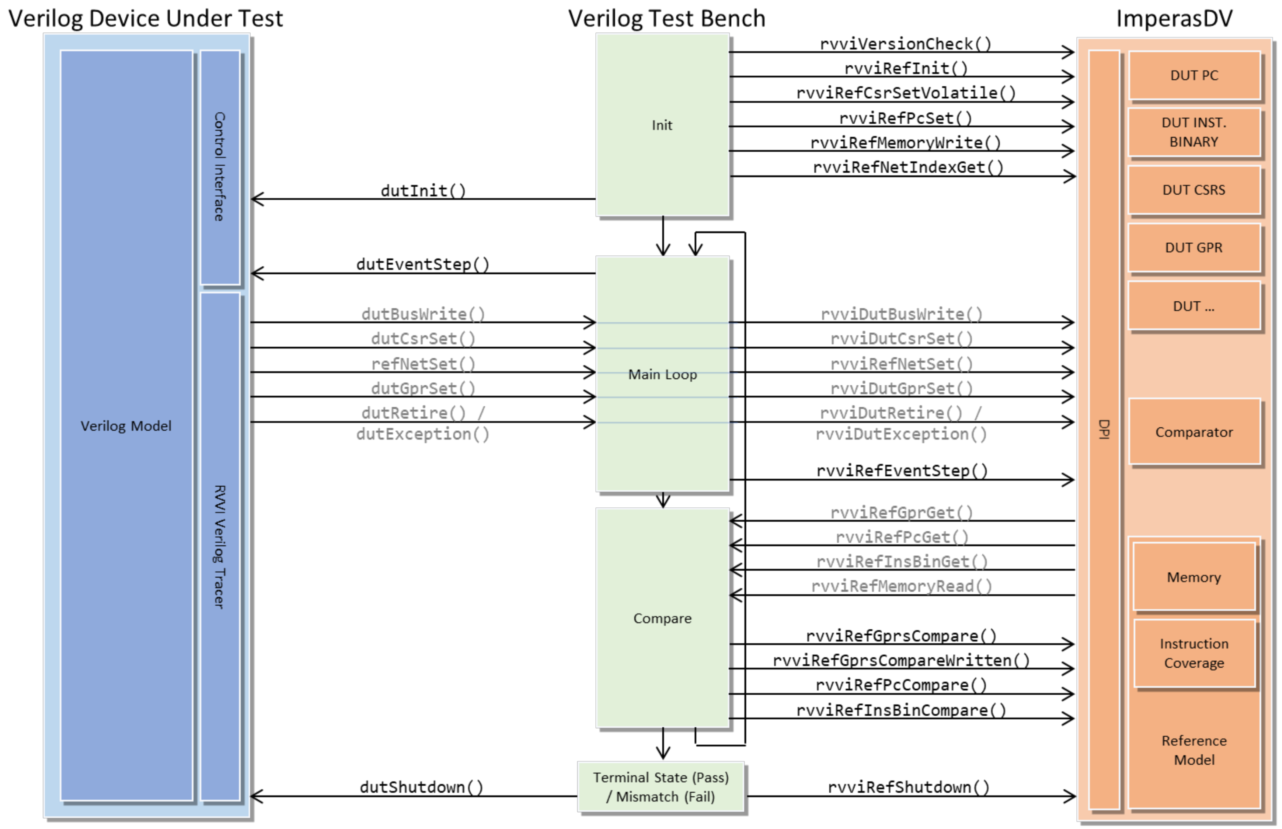
DV Functions



- Select model, use variant, configure
- Reference model encapsulation
  - Enable instruction coverage
- Includes DUT reference state storage
- Includes synchronization technology
  - Can run sync, async, interrupts, debug, multi-hart
- Includes comparison technology
  - Comparisons are done on *instruction retirement*; enables DV of multi-issue and OoO pipeline processors
- Can be used in C/C++ or SystemVerilog test bench / harness
  - Uses RVVI-API
- Very simple to use – the 'smarts' are built-in



ImperasDV Setup



- Reference model setup
- Configuration of register and memory initialization
- Selection of what to compare (depends on DUT tracer capabilities)
  - PC, GPR, CSR, FPR, VR, decode, net, hart …
- Select capabilities:
  - sync-step-compare or async-step-compare
- Trace and logging set up
- Selection of built-in instruction coverage
- Choice of DV control options

Summary

- RISC-V processor developers need to do comprehensive verification of the RTL implementation
- Processor DV methodology has been evolved by Imperas, together with customers and partners
- Asynchronous step-compare methodology provides the most comprehensive, most efficient RISC-V DV flow
- Key technologies include the Imperas OVP reference model and the ImperasDV verification IP

Thank you!

LarryL@imperas.com